20 VIDEOLEZIONI DI BASIC
PER IMPARARE COL VIC 20

GRUPPO
EDITORIALE
JACKSON

Il plotter
I sottoprogrammi
GOSUB RETURN
ON ... GOSUB
BUBBLE SORT
SHELL SORT
Programmare con le subroutine
Videogioco n° 13

13

COMMODORE
VIC 20

**VIDEO BASIC VIC 20**
Pubblicazione quattordicinale
edita dal Gruppo Editoriale Jackson
**Direttore Responsabile:**
Giampietro Zanga
**Direttore e Coordinatore**
**Editoriale:** Roberto Pancaldi
**Autore:** Softidea - Via Indipendenza 88 - Como
**Redazione software:**
Francesco Franceschini, Enrico Braglia,
Fabio Calanca
**Segretaria di Redazione:**
Marta Menegardo
**Progetto grafico:**
Studio Nuovaidea - Via Longhi 16 - Milano
**Impaginazione:**
Silvana Corbelli
**Illustrazioni:**
Cinzia Ferrari, Silvano Scolari
**Fotografie:**
Marcello Longhini
**Distribuzione:** SODIP
Via Zuretti, 12 - Milano
**Fotocomposizione:** Lineacomp S.r.l.
Via Rosellini, 12 - Milano
**Stampa:** Grafika '78
Via Trieste, 20 - Pioltello (MI)
**Direzione e Redazione:**
Via Rosellini, 12 - 20124 Milano
Tel. 02/6880951/5


Autorizzazione alla pubblicazione Tribunale di Milano n° 422 del 22-9-1984
Spedizione in abbonamento postale Gruppo II/70 (autorizzazione della Direzione Provinciale delle PPTT di Milano).
Prezzo del fascicolo L. 8.000
Abbonamento comprensivo di 5 raccoglitori L. 165.000
I versamenti vanno indirizzati a: Gruppo Editoriale Jackson S.r.l. - Via Rosellini, 12 20124 Milano, mediante emissione di assegno bancario o cartolina vaglia oppure utilizzando il c.c.p. n° 11666203.
I numeri arretrati possono essere richiesti direttamente all'editore inviando L. 10.000 cdu. mediante assegno bancario o vaglia postale o francobolli.
Non vengono effettuate spedizioni contrassegno.

**Gruppo Editoriale Jackson**

# SOMMARIO



# Introduzione

*Ancora grafica, parliamo infatti del plotter: il "tecnigrafo" del tuo computer. E poi ... un argomento fondamentale per ogni buon programmatore: i sottoprogrammi. Il loro uso consente infatti di suddividere un programma impegnativo in piccole parti maneggevoli e facili da comprendere con l'ulteriore vantaggio di risparmiare memoria. Un risparmio di tempo e lavoro è invece offerto dal sort (ordinamento) che permette di avere i dati in un comodo ordine alfabetico o numerico.*

## Il plotter

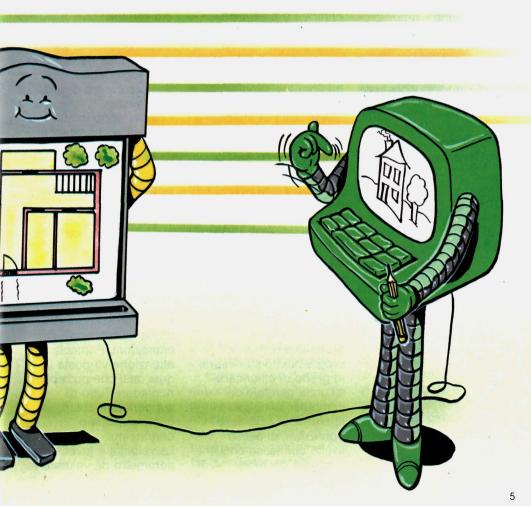
Lo schermo video costituisce la principale periferica di output dei personal computer, ma non la sola. Abbiamo già parlato delle cosiddette stampanti grafiche, periferiche che consentono di riprodurre sulla carta - a volte addirittura a colori - tutto quanto si trova sullo schermo in un certo istante.

Una stampante presenta tuttavia nel funzionamento grafico una serie di inconvenienti e svantaggi, che, per determinati utilizzi od in particolari circostanze, possono renderla inadeguata.
Le immagini prodotte da una stampante, non consentono mai di ottenere definizioni al di sopra di un certo livello, essendo costituite da singoli punti affiancati gli uni accanto agli altri. Le linee (rette o curve che siano) sono quindi sempre caratterizzate - in modo più o meno accentuato - da discontinuità e

scostamenti di vario genere rispetto alla traiettoria "ideale", eseguibile a tratto continuo.
Quando è necessario ottenere copie su carta con precisione e definizioni superiori, le stampanti grafiche devono quindi cedere il passo ai loro "fratelli maggiori", i plotter.
I plotter sono infatti periferiche progettate e costruite esplicitamente per disegnare; possiedono quindi requisiti e prestazioni grafiche nettamente migliori, qualitativamente superiori anche agli schermi video.

## Come funziona un plotter

Il principio-base del funzionamento di un plotter è abbastanza semplice: il foglio è fisso su un piano e su di esso scorre un braccio (molto simile a quello di un normale tecnigrafo) mosso da due motori di precisione, che ne controllano il posizionamento orizzontale e verticale.
Il braccio trasporta nel proprio moto una penna, che durante il movimento può essere alzata o abbassata a contatto del foglio; la composizione dei due spostamenti consente allora alla penna di raggiungere qualunque punto del piano e di generare qualsiasi figura.
Esistono tuttavia anche altri modelli di plotter, oltre a quello appena visto (che prende il nome di plotter "ad equipaggio mobile"): i plotter a rullo e i plotter a tamburo.
Il tipo a rullo prevede che il foglio venga fissato su un rullo, o su un supporto teso tra due rulli, in modo che il movimento lungo una direzione sia dato dalla rotazione del rullo e l'altro dallo spostamento, limitato a quella direzione, della penna.
Questo tipo di plotter funziona, per intenderci, in modo molto simile agli elettrocardiografi: la carta rotola in un senso

e il pennino può muoversi nell'altra direzione, perpendicolare alla prima.
Per il tipo a tamburo la differenza dal precedente consiste nel fatto che il foglio non è completamente fissato a un supporto, ma viene "afferrato" da un dispositivo di trascinamento, che lo fa scorrere avanti ed indietro, esattamente come nel caso precedente.
La differenza di prestazioni dei tre tipi si verifica rispetto alla accelerazione che la penna può raggiungere: essendo il peso delle parti in movimento molto minore nei plotter ad equipaggio mobile, è

chiaro che in questo modello la velocità di spostamento e di tracciamento risulta molto maggiore rispetto agli altri due.
Le linee tracciate da un plotter, a differenza di quelle rappresentate su un video a pixel o su una stampante ad aghi, appaiono continue: la ragione è dovuta al fatto che la penna - quando scorre sopra il foglio disegnando qualcosa - non viene mai sollevata dalla carta, lasciando quindi un tratto continuo.
Il movimento della penna del plotter è perciò variabile con continuità: in altre parole, il plotter è tipicamente un dispositivo analogico, anche se ormai esistono plotter quasi eclusivamente digitali.
Poiché il computer può produrre solo numeri in forma digitale, la connessione tra unità centrale e plotter deve quindi avvenire attraverso un convertitore digitale-analogico, in grado di trasformare i segnali di comando per i motori - inviati dalla CPU sotto forma di informazioni binarie - in grandezze analogiche accettabili dai motori.
Tale problema non si presenta nei plotter digitali, che richiedono tuttavia - a causa della complessità del lavoro da svolgere - la presenza di un microprocessore dedicato (cioè di un vero e proprio microcalcolatore progettato e programmato su misura), in grado di affiancarsi all'unità centrale del computer principale, liberandola così dalle operazioni di basso livello connesse con il controllo dei motori.

## Scegliere un plotter

I parametri interessanti per i plotter sono numerosi. Il formato del foglio, per esempio: alcuni plotter accettano - se necessario - formati minori dello standard, adeguando tutto il disegno alle nuove dimensioni. Tale prestazione risulta comunque inutilmente dispendiosa nei casi in cui i fogli debbano sempre essere di una sola misura.
I plotter sono infatti dispositivi abbastanza costosi (in genere molto più delle stampanti) ed ogni "optional" in più non fa altro che aumentarne ulteriormente il prezzo.
Un secondo parametro è il tipo di supporto ammesso, ossia quali e quanti tipi di carta o altro materiale possono essere utilizzati senza che le prestazioni del plotter subiscano alterazioni. Particolare attenzione va posta in quei casi che potrebbero richiedere l'uso di supporti particolari, come plastica o cartoncino.
La velocità è invece un parametro da valutare in

riferimento a diverse misurazioni possibili: quando si traccia una linea lungo una diagonale, la velocità risulta infatti maggiore che lungo uno degli assi, essendo composta dalla somma delle due componenti, orizzontale e verticale, che pongono in movimento il braccio. Inoltre la maggioranza dei plotter ha velocità diverse, a seconda che la penna sia alzata od abbassata (naturalmente maggiore nel primo caso). I valori limite di velocità per i plotter più diffusi vanno comunque da meno di 10 centimetri al secondo a quasi un metro al secondo per quelli più sofisticati.
Un altro importante parametro è la risoluzione. Per i plotter si considerano due tipi di risoluzione: meccanica ed indirizzabile. La risoluzione meccanica è data dal minimo movimento della penna ed è logicamente dipendente dai meccanismi e dai motori adibiti a tale compito. La risoluzione indirizzabile è invece data dal minimo movimento che si può provocare con una istruzione via software. Perciò la risoluzione effettivamente utilizzabile è quella indirizzabile che, - al massimo - può essere pari a quella meccanica.
Un altro parametro interessante è la cosiddetta ripetibilità e consiste nella capacità di tornare su un punto, già toccato in precedenza, con il minore scarto possibile. Una classica prova di ripetibilità consiste nel far disegnare cerchi, figure piane e rette parallele, magari con cambi di penna.
Alcuni plotter, infatti, permettono il cambio automatico del colore o dello spessore della penna, prelevando la penna prescelta da un apposito portapenne, che di solito si trova sul bordo del plotter stesso, nel quale la penna viene depositata dopo l'uso.
Al momento dell'acquisto possono comunque venire richiesti altri parametri, dipendenti dalla specifica applicazione a cui il plotter sarà destinato (per esempio la silenziosità, l'ingombro o il riconoscimento di particolari istruzioni grafiche).
Un plotter, nel complesso, risulta comunque una periferica tuttora molto sofisticata, adatta più per usi professionali che hobbystici.
Per un uso normale - e spesso anche oltre - una "semplice" stampante grafica è certamente la soluzione migliore (quanto meno dal punto di vista economico), permettendo in qualunque circostanza prestazioni di tutto rispetto ed affidabilità di prim'ordine.

## Sottoprogrammi

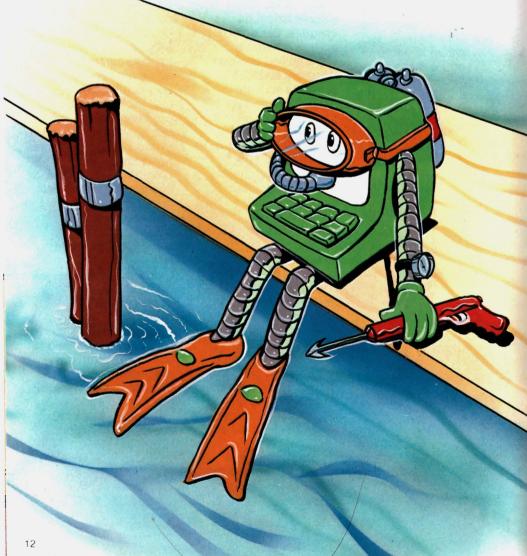
Risulta spesso assai utile (per non dire indispensabile) avere la possibilità di poter ripetere più volte, una stessa operazione od uno stesso gruppo di operazioni in punti diversi del programma o in condizioni leggermente differenti.
Finora le eventualità di questo tipo le abbiamo sempre risolte scrivendo le medesime linee nei vari punti del programma, ripetendo cioè tutte le istruzioni che di volta in volta si rendevano necessarie per portare a termine il compito che ci eravamo prefissi.
Una simile soluzione presenta però tutta una serie di inconvenienti e svantaggi che un buon programmatore non può non tenere in debita considerazione: l'allungamento del programma (con conseguente inutile occupazione della memoria), la maggiore possibilità di errori di battitura (con ulteriori difficoltà nella modifica e nella variazione delle varie istruzioni in una successiva fase di messa a punto del programma), la scarsa corrispondenza tra il modo di ragionare a cui siamo di solito abituati e il modo di operare che simili condizioni impongono al computer e - non ultima - la minore facilità di lettura.
Questi problemi si aggravano inoltre con l'aumentare delle dimensioni del programma stesso, arrivando in alcuni casi ad essere così rilevanti da riuscire a compromettere anche lavori che inizialmente sembrano essere impostati nel migliore dei modi.
Un metodo estremamente efficace per evitare tutti questi inconvenienti può allora essere quello di ricorrere all'utilizzo dei sottoprogrammi.
Un sottoprogramma (molte volte chiamato anche subroutine) non è altro che un gruppo di istruzioni che il programma può richiamare ed adoperare in qualunque momento dell'esecuzione.

L'uso delle subroutine consente quindi di scrivere separatamente quelle parti del programma che devono essere *ripetute più volte*, permettendo così al programmatore di suddividere il problema in blocchi di istruzioni elementari, ciascuno dei quali con dei risultati in ingresso e dei risultati in uscita.
La parte *"sotto"* del nome indica infatti che i sottoprogrammi operano sotto il controllo del programma principale, cioè ai livelli più bassi dell'esecuzione (ma non per questo meno importanti).

La parte "routine" del nome indica invece che le subroutine vengono *spesso utilizzate* per eseguire diverse volte in uno stesso programma dei processi ripetitivi, cioè di routine. Sotto questo aspetto i sottoprogrammi sono quindi molto simili alle funzioni, in quanto adempiono a uno specifico compito, al

quale è possibile riferirsi più volte ed in qualunque punto di uno stesso programma.
Tuttavia, il grosso vantaggio dei sottoprogrammi è che essi - a differenza delle funzioni - non fanno parte integrante del linguaggio BASIC permanentemente inserito all'interno della memoria ROM: ogni volta occorre infatti fornirne al computer la esatta definizione, specificando quindi le singole azioni da eseguire, per poterle portare a termine.
Ciò significa che con istruzioni BASIC è possibile scrivere sottoprogrammi per risolvere qualsiasi problema, indipendentemente dalle possibilità o dalle prestazioni offerte dal singolo elaboratore.
Così come un libro risulta molto più semplice da leggere (e, per l'autore, da *modificare) quanto più* numerosi sono i capitoli che lo compongono, anche la struttura di un programma può allora essere resa più chiara ed ordinata, spezzandone il corpo principale in più parti, incluse come subroutine (ciascuna delle quali magari scritta e provata separatamente dalle altre).
Un programma può avere inoltre una vita alquanto incerta: lo si può far funzionare ripetutamente, facendogli sempre svolgere la stessa funzione, o lo si può modificare per adattarlo a nuove e diverse necessità.
Per scrivere in modo chiaro ed intelligente i programmi occorre pertanto cercare di facilitare al massimo la possibilità di eventuali adattamenti e migliorie.
I sottoprogrammi sono quindi universalmente conosciuti come i cavalli vincenti dei migliori programmatori: il loro uso costituisce infatti un ottimo biglietto da visita per qualunque programma, permettendo una presentazione del lavoro più pulita, leggibile, ben organizzata e, soprattutto, facile da *modificare.*
Cerca quindi di impararne, oltre che la pura e semplice definizione "grammaticale", anche (e soprattutto) la utilità e la praticità di utilizzo: avrai raggiunto, con poca e ben spesa fatica, un ulteriore ed importante traguardo sulla strada della programmazione.

## GOSUB RETURN

In BASIC l'uso dei sottoprogrammi è estremamente semplice: basta infatti specificare il numero di linea dal quale il sottoprogramma è stato fatto iniziare. Detto così, sembrerebbe pertanto molto naturale ricorrere a una normale istruzione GOTO.
Il salto dal programma principale al sottoprogramma (detto anche "chiamata" della subroutine) deve invece

avvenire attraverso un breve e particolare comando: GOSUB (*abbreviazione di GO to* SUBroutine).
GOSUB permette difatti di passare l'esecuzione dal corpo principale del programma al gruppo di istruzioni che costituiscono un certo sottoprogramma, *in* modo molto simile a quanto avviene quando utilizzi una normale istruzione di salto (GOTO) per trasferire il controllo ad un'altra linea del programma.
Esiste tuttavia una sostanziale ed importantissima differenza tra le istruzioni GOSUB e GOTO: GOSUB effettua infatti il salto di partenza proprio come una GOTO, ma in aggiunta ricorda anche il punto di partenza da cui tale salto ha avuto luogo.
Spieghiamoci meglio: l'istruzione

```
30 GOSUB 1000
```

trasferisce il controllo del programma alla linea numero 1000 (che rappresenta quindi l'inizio di un *sottoprogramma*), memorizzando però nello stesso momento - in una zona della memoria RAM appositamente riservata - anche il numero della linea dalla quale il comando di salto è partito (cioè 30).
L'esecuzione prosegue allora dalla linea 1000 in avanti, fino al momento in cui un'altra istruzione segnerà la fine della *subroutine. In* BASIC questa istruzione è RETURN (il cui significato - RITORNA - non lascia spazio ad alcun dubbio).
Quando l'interprete BASIC incontra il comando RETURN, il controllo viene infatti riportato all'istruzione immediatamente seguente la GOSUB eseguita in precedenza, nel nostro caso alla linea 30, *riprendendo così la* sequenza che il salto alla subroutine aveva interrotto.
GOSUB e RETURN devono pertanto essere sempre adoperate in coppia: la prima per effettuare la chiamata alla subroutine, la seconda per ritornare al punto di partenza.
La mancanza dell'istruzione RETURN alla fine di un *sottoprogramma provoca* infatti una grossa confusione nella memoria del computer, con il rischio di incorrere talvolta anche nella segnalazione di un messaggio di errore.
Altrettanto pericoloso è tentare di inserire un RETURN di troppo: se il tuo VIC 20 incontra

RETURN senza aver prima incontrato il corrispondente comando GOSUB, l'unica cosa che può fare - visto che non sa dove continuare l'esecuzione - è infatti quella di avvertirti dell'errore, visualizzando il messaggio

```
RETURN WITHOUT
GOSUB
```

Per verificarlo è sufficiente che tu scriva ed esegua

```
10 RETURN
```

Il messaggio di errore comparirà immediatamente.
In un programma possono comunque trovare posto tutte le subroutine di cui puoi aver bisogno; basta soltanto che ciascuna di esse disponga del proprio RETURN, che ne delimiterà quindi la fine.
All'interno di un sottoprogramma è inoltre perfettamente lecito inserire altre istruzioni GOSUB, che, a loro volta, possono richiamare altre subroutine, e così via: in questo caso si parla di sottoprogrammi nidificati, cioè posti uno dentro l'altro:

```
 10 GOSUB 100
100 ........
110 GOSUB 450
    ........
185 RETURN
450 .......
530 RETURN
```

Occorre tuttavia fare un minimo di attenzione a non esagerare troppo nell'inserimento di sottoprogrammi nidificati, dal momento che il calcolatore potrebbe non riuscire a prendere nota di tutte le chiamate.
Abbiamo infatti detto che l'istruzione GOSUB, oltre al salto vero e proprio verso il sottoprogramma, memorizza anche il numero di linea del GOSUB chiamante in un'area della memoria opportunamente predisposta, chiamata anche STACK (catasta).
Quest'ultima può essere immaginata come una pila di fogli sovrapposti, in ognuno dei quali viene scritta la riga da cui dovrà riprendere l'esecuzione dopo ciascun RETURN.
Ad ogni GOSUB l'interprete BASIC aggiunge allora un nuovo foglio, mentre ad ogni RETURN ne toglie uno. Ad un certo punto può però succedere che non vi siano più disponibili altri fogli di carta per scrivere il numero di linea: in altre parole, lo STACK risulta interamente occupato.
In questo caso si avrà la segnalazione di un messaggio di errore per mancanza di memoria disponibile. Tale evenienza, a parte casi particolari e proprio per questo molto rari, è tuttavia solitamente imputabile alla presenza

# LINGUAGGIO

di errori nel programma (ad esempio l'uso inappropriato di istruzioni GOTO o di cicli FOR ... NEXT all'interno di sottoprogrammi può riempire con inutile facilità la catasta del tuo VIC 20).
Eccoti allora subito un (cattivo) esempio, che ti illustra come può accadere questo errore:

```
10 LET K = 0
20 LET K = K + 1
30 PRINT K
40 GOSUB 20
50 RETURN
```

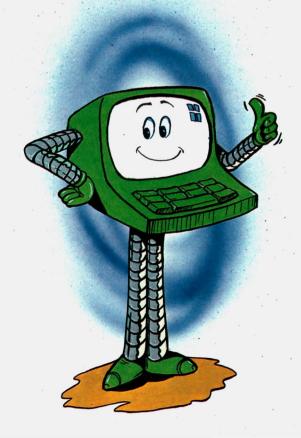
Eseguendolo, vedrai comparire sullo schermo i successivi valori assunti dalla variabile K, corrispondenti al numero di chiamate della subroutine 20: ad un certo punto lo STACK risulterà completamente occupato (visto che la RETURN alla linea 50 non viene mai eseguita) ed il calcolatore sarà costretto a bloccarsi. L'ultimo valore presente sullo schermo video ti indicherà la dimensione dello STACK e - di conseguenza - anche quante subroutine possono essere contemporaneamente chiamate sul VIC 20.
Il programma che segue ti illustra invece il vantaggio che presenta l'utilizzo dei comandi GOSUB e RETURN per controllare la gestione di un sottoprogramma.
Supponiamo di voler fare in modo che il nostro computer - una volta accettate in ingresso due stringhe di caratteri qualsiasi - visualizzi tali stringhe, "incorniciandole" rispettivamente con asterischi (*) e con caratteri dollaro ($).
Vediamo innanzi tutto come potremmo fare senza subroutine:

```
 10 INPUT A$, B$
 20 FOR I = 1 TO LEN (A$) + 4
 30 PRINT "*";
 40 NEXT I
 50 PRINT
 60 PRINT "*"; A$; "*"
 70 FOR I = 1 TO LEN (A$) + 4
 80 PRINT "*";
 90 NEXT I
100 PRINT : PRINT
110 FOR I = 1 TO LEN (B$) + 4
120 PRINT "$";
130 NEXT I
140 PRINT
150 PRINT "$"; B$; "$"
160 FOR I = 1 TO LEN (B$) + 4
170 PRINT "$";
180 NEXT I
```

Usando una subroutine avremmo invece potuto scrivere:

```
 10 INPUT A$, B$
 20 LET C$ = A$ : LET D$ = "*"
 30 GOSUB 500
 40 PRINT D$; "   "; C$; "   "; D$
 50 GOSUB 500
 60 PRINT
 70 LET C$ = B$ : LET B$ = "$"
 80 GOSUB 500
 90 PRINT D$; "   "; C$; "   "; D$
100 GOSUB 500
110 END
500 FOR I = 1 TO LEN (C$) + 1
510 PRINT D$;
520 NEXT I
530 PRINT
540 RETURN
```

Adesso che abbiamo scritto i due programmi possiamo cominciare a rilevare un certo numero di considerazioni che soltanto grazie ad un esempio applicativo è possibile mettere nella giusta evidenza e che - ad una prima occhiata - possono sfuggire o sembrare banali.
Innanzi tutto il secondo programma è più breve del primo, ma questo - se va a spese della chiarezza e della leggibilità -, può non essere assolutamente un vantaggio (anche se non è il nostro caso).
Una eventuale modifica, poi, risulta molto più scomoda da effettuare nella prima versione: se per esempio volessimo incorniciare la prima stringa di punti esclamativi, anziché di asterischi, sarebbe necessario scorrere tutto il programma alla ricerca delle istruzioni che vanno cambiate (oltretutto, nonostante la semplicità della modifica, risultano ben tre linee da correggere: la 30, la 60 e la 80). Quest'ultima è un'operazione che - soprattutto se i programmi iniziano ad assumere dimensioni di un certo rilievo - risulta estremamente lunga e noiosa. Inoltre, con questo sistema, esiste una maggiore probabilità di commettere eventuali errori di battitura, a tutto svantaggio della sicurezza e della affidabilità del programma: può infatti succedere di non notare l'inconveniente (che magari capita solo in una certa situazione) e credere che il programma funzioni senza problemi.
Nel programma con la subroutine, invece, è sufficiente cambiare in punto esclamativo l'asterisco alla linea 20 ed il gioco è subito fatto.
Un'altra cosa da notare

è l'introduzione, nella seconda versione, delle due nuove variabili C$ e D$. La loro funzione è tanto semplice quanto utile, e viene spesso messa a frutto in numerosi programmi.
In molti casi, infatti, uno stesso lavoro deve essere fatto più volte, ma in modo leggermente diverso.
Nel nostro programma, per esempio, occorre stampare due file di caratteri (asterischi e dollari), basandosi sulle lunghezze di due distinte stringhe (A$ e B$).
Affinché la subroutine possa svolgere questo compito sono necessarie due informazioni: il carattere da stampare e la stringa di cui si deve calcolare la lunghezza.
Queste informazioni, che il programma principale passa alla nostra *subroutine, sono dette* "parametri di ingresso" della subroutine. Poiché il sottoprogramma alla linea 500 usa come parametri di ingresso le variabili C$ e D$, occorre che il programma principale - prima di chiamare la subroutine - prepari in C$ e D$ i valori con cui dovrà successivamente operare.

Arriviamo infine alle ultime due considerazioni che, per quanto all'apparenza banali, possono in numerose circostanze evitare contrattempi ed errori apparentemente inspiegabili.
La prima riguarda la collocazione delle subroutine: è sempre bene metterle in fondo al programma principale, *numerandole - come nel* nostro caso - in maniera che risaltino immediatamente alla prima occhiata. In casi di correzione, questo ti eviterà sicuramente errori, incertezze e perdite di tempo.
La seconda osservazione riguarda invece l'istruzione END che, come puoi vedere, è stata messa soltanto nella seconda versione: nella prima - nonostante fosse perfettamente lecito - ha potuto essere trascurata, dal momento che, quale ultima istruzione si poteva tralasciare.
Prova adesso a toglierla alla seconda versione e ad eseguire il programma: vedrai apparire sullo schermo il messaggio di errore RETURN WITHOUT GOSUB.
Ciò è dovuto al fatto che *il VIC 20 non sa dove* termina il programma vero e proprio; invece di "capire" che l'esecuzione va arrestata subito dopo la linea 100, esso prosegue nelle linee successive, trattandole come se facessero parte del corpo principale e non della subroutine.
Il RETURN alla riga 540, non essendo stato chiamato da alcun GOSUB, provoca quindi l'insorgere dell'errore.
Il modo migliore per separare il programma principale dalle subroutine è allora quello di inserire l'istruzione END: in questa maniera ci si mette sicuramente al riparo da pericolose eventualità di errori od arresti indesiderati.

## Sintassi dell'istruzione GOSUB

GOSUB numero linea

## Sintassi del comando RETURN

RETUR

## ON ... GOSUB

Analogamente a ON ... GOTO, l'istruzione ON <espressione> GOSUB consente di passare il controllo dell'esecuzione ad una qualsiasi delle subroutine presenti nel programma, in funzione del valore assunto dall'espressione numerica compresa tra le parole ON e GOSUB. ON ... GOSUB consente, pertanto, la realizzazione di una struttura alternativa multipla; vediamo insieme un esempio:

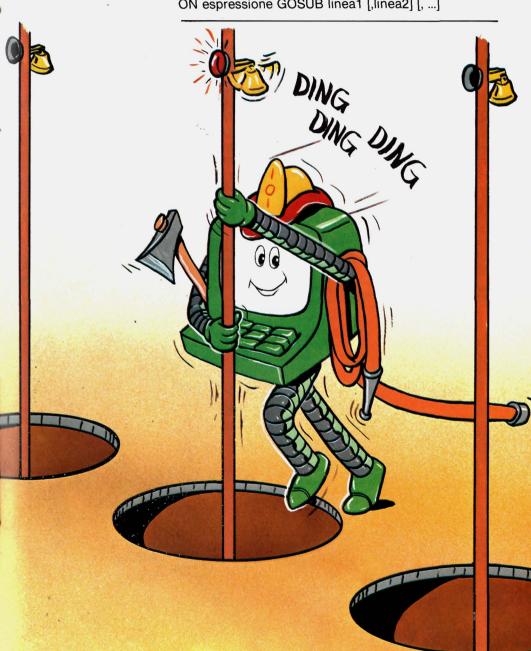
```
10 INPUT A
20 ON A GOSUB 40, 50, 60, 70
30 GOTO 10
40 PRINT "HAI PREMUTO 1" : RETURN
50 PRINT "HAI PREMUTO 2" : RETURN
60 PRINT "HAI PREMUTO 3" : RETURN
70 PRINT "HAI PREMUTO 4" : RETURN
```

Il programma inizia richiedendo in ingresso un valore numerico. Valuta quindi l'espressione compresa tra ON e GOSUB (nel nostro caso semplicemente costituita da A) e - a seconda che la parte intera di A abbia valore 1, 2, 3, 4 - esegue la procedura indicata dal primo, dal secondo dal terzo o dal quarto numero di linea presente dopo la parola GOSUB. Se il valore di A è invece zero, oppure supera la quantità di numeri di linea presenti nell'istruzione, il programma prosegue con la linea di programma immediatamente seguente. Proprio questa è la ragione per cui è stata inserita la riga 30: essa provocherà infatti la riesecuzione del programma fintato che dalla tastiera sarà stato battuto un valore compreso tra 1 e 4. Un valore negativo dell'espressione provocherà invece il messaggio:

```
? ILLEGAL QUANTITY
```

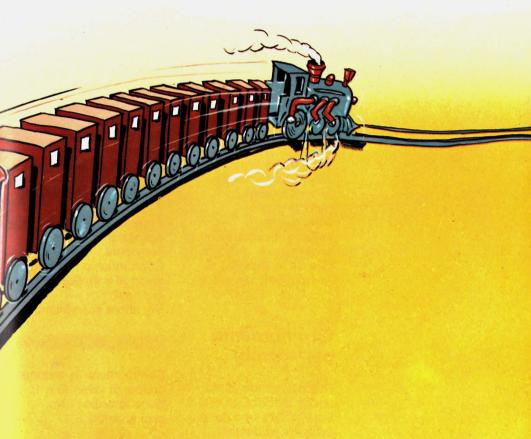
L'istruzione ON ... GOSUB può essere utile solo quando le differenti possibilità alle quali sei interessato hanno valori consecutivi, per di più inizianti con il valore uno, oppure siano riducibili a tale situazione.

## Sintassi dell'istruzione

ON espressione GOSUB linea1 [,linea2] [, ...]

## Sort

Nella elaborazione dei dati risulta spesso necessario ordinare e classificare le varie informazioni (fornite in ingresso o raggiunte come risultato di elaborazioni) secondo determinati criteri o necessità.
Normalmente, infatti, le informazioni vengono inserite nel computer in un ordine più o meno "sparso", mentre in uscita vengono richieste in modo che risultino ordinate secondo categorie o relazioni ben precise, per esempio *alfabetiche o numeriche.*
Con il termine "sort" si intendono allora tutte quelle tecniche, di funzionamento più o meno raffinato, che hanno come unico scopo quello di sistemare i dati secondo un certo ordine.
Quando si devono ordinare centinaia o migliaia di dati è fondamentale disporre di algoritmi veloci ed efficienti. All'aumentare del numero di informazioni da ordinare (che di solito vengono inserite nella memoria dell'elaboratore come elementi di array) il numero di operazioni richieste per l'ordinamento si allunga infatti in maniera a volte drammatica.

# PROGRAMMAZIONE

Per quanto all'argomento siano dedicati interi libri (adatti soprattutto in vista di un utilizzo professionale dell'elaboratore), vedremo comunque che esistono alcuni metodi di sort in grado di soddisfare ottimamente le normali esigenze che può avere un utilizzatore di personal computer.
Prima di passare a vedere i programmi veri e propri è però necessario fare una piccola premessa.
Per effettuare un ordinamento occorre infatti per prima cosa stabilire una relazione di precedenza tra gli elementi da ordinare.
Occorre, in altre parole, stabilire quale viene prima e quale dopo.
Se per i numeri non ci sono problemi, il concetto di ordine alfabetico non è poi così chiaro: ci sono infatti stringhe composte da

lettere, numeri e simboli vari. A meno di casi particolari, il sort si effettua di solito in ordine di codice ASCII. Gli elementi vengono cioè confrontati carattere per carattere: quello con il codice ASCII minore viene considerato precedente l'altro.
Così la stringa "CARRO" deve venire dopo la stringa "CARRI" (il codice ASCII di "I" è minore di quello della "O"), mentre "CLASSE 1960" precede "CLASSE DI FERRO" (il codice di "1" precede il codice di "D").
Quando il calcolatore incontrerà quindi due stringhe di caratteri come GATTO e CANE, considererà CANE "minore" di GATTO, visto che il codice ASCII del carattere C risulta inferiore di quello del carattere G.
Un eventuale ordinamento alfabetico potrà allora essere effettuato sfruttando proprio questa caratteristica.
Fatta questa precisazione passiamo a vedere più in dettaglio due tra le tecniche di ordinamento più diffuse.

## Ordinamento per scambi

Questo metodo (detto anche BUBBLE SORT) è sicuramente uno dei più semplici e comuni: presenta il vantaggio di non richiedere molta memoria per essere utilizzato, ma - di contro - non lavora con estrema velocità.
Il metodo consiste nel confrontare, uno dopo l'altro ed a coppie, gli elementi che compongono un array. Se qualche confronto rivela che gli elementi non sono ordinati, essi vengono scambiati e il confronto continua con gli elementi successivi. Quando l'array sarà stato scandito in tutta la sua lunghezza senza che vi siano stati scambi l'ordinamento sarà stato completato.
Consideriamo per esempio i 5 valori, disposti casualmente:

6 4 1 8 0

e supponiamo di volerli collocare in ordine crescente.
*Confrontiamo i primi due* valori (6 e 4). Sono ordinati?
No, allora scambiamoli:

4 6 1 8 0

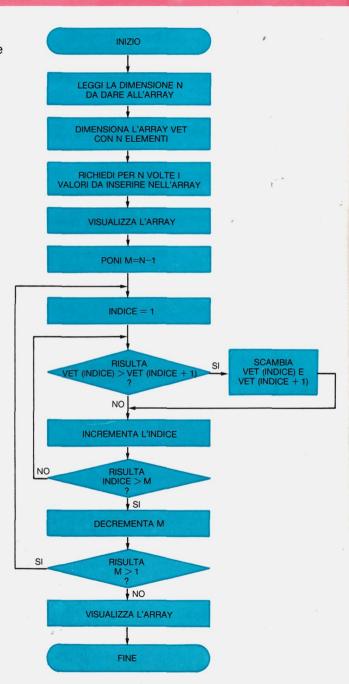
Confrontiamo la seconda coppia di valori (6 e 1). Sono in ordine? No, scambiamoli:

4 1 6 8 0

Confrontiamo la terza coppia (6 e 8). Sono ordinati? Sì, allora proseguiamo, lasciandoli invariati.
Passiamo all'ultima coppia (8 e 0). Visto che non sono ordinati, scambiamoli tra loro:

4 1 6 0 8

I numeri sono più ordinati di prima, ma non è ancora stato raggiunto l'ordinamento completo.
Possiamo comunque notare che almeno il valore 8 ha già raggiunto la posizione corretta (cioè l'ultima): il nome bubble sort - letteralmente "ordinamento a bolle" - deriva infatti dalla constatazione che ciascun elemento avanza gradatamente verso la posizione che gli compete, proprio come fa una colonna di bolle d'aria spostandosi nell'acqua.
Potremo pertanto ricominciare da capo l'intero processo, tenendo però conto che stavolta si può concludere alla terza coppia, visto che l'ultimo valore è già al posto giusto. Proseguendo in questo modo, uno dopo l'altro, gli elementi verranno ordinati secondo i valori crescenti, proprio come volevamo inizialmente.
Puoi vedere lo schema a blocchi dell'intero

procedimento illustrato qui a fianco: esso include anche la parte relativa al dimensionamento del vettore da ordinare e all'inserimento dei vari elementi, oltre naturalmente alla visualizzazione dell'array.

Ed ecco il corrispondente listato BASIC

```
10 INPUT "QUANTI ELEMENTI";N
20 DIM VET(N)
30 FOR I=0 TO N
40 INPUT VET(I)
50 NEXT I
60 GOSUB 500:REM VISUALIZZA L'ARRAY
   NON ORDINATO
70 GOSUB 1000:REM ORDINA L'ARRAY
80 GOSUB 500:REM VISUALIZZA L'ARRAY
   ORDINATO
90 END
500 FOR I=0 TO N
510 PRINT VET(I);" ";
520 NEXT I
530 PRINT:PRINT
540 RETURN
1000 FOR M=N-1 TO 0 STEP -1
1010 FOR INDICE=0 TO M
1020 IF VET(INDICE)>VET(INDICE+1) THEN
     GOSUB 2000
1030 NEXT INDICE
1040 NEXT M
1050 RETURN
2000 LET I=VET(INDICE)
2010 LET VET(INDICE)=VET(INDICE+1)
2020 VET(INDICE+1)=I
2030 RETURN
```

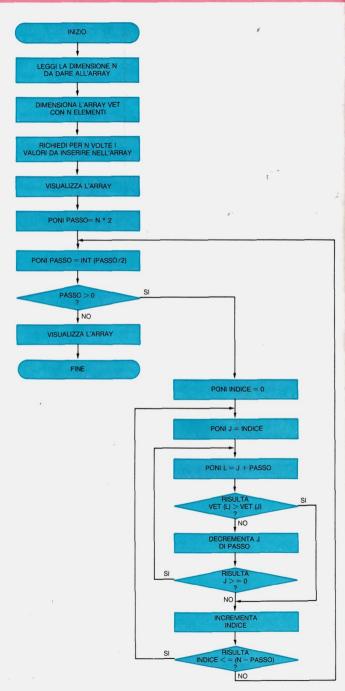
Nel caso tu volessi evitare la fatica di inserire a mano i numeri dell'array (la cosa può diventare noiosa, se sono molti) è sufficiente che tu sostituisca la linea 40 con

```
40 LET VET(I) = INT (RND (0) *5000)
```

Eseguendo il programma, ti accorgerai che per l'ordinamento viene richiesto un tempo piuttosto lungo, specialmente quando gli elementi cominciano ad essere abbastanza numerosi. Infatti, mentre per l'ordinamento di 10 dati il programma richiede 45 confronti tra le varie coppie di numeri, con 250 dati in ingresso il numero dei confronti da effettuare sale a ben 31125!

## Shell sort

Come vedrai, questo secondo metodo di ordinamento (chiamato anche suddivisione binaria) ha il vantaggio di risultare più veloce del precedente, pur richiedendo circa lo stesso numero di istruzioni.
Il SHELL SORT è in pratica costituito da una serie di bubble sort. Gli elementi del vettore, anziché essere direttamente confrontati secondo coppie adiacenti, vengono *preordinati con una serie* di bubble sort: il primo di passo pari a N (chiamando con N il

numero di elementi dell'array, confrontando cioè il primo termine con l'ultimo), il secondo con passo N/2, il terzo N/4 e così via. Alla fine, a forza di dividere, si arriva al passo 1 e si procede con il bubble sort visto in precedenza.
Ecco lo schema a blocchi corrispondente:

Ed ecco il listato BASIC:

```
  10 INPUT "QUANTI ELEMENTI";N
  20 DIM VET(N)
  30 FOR I=0 TO N
  40 INPUT VET(I)
  50 NEXT I
  60 GOSUB 500:REM VISUALIZZA L'ARRAY
     NON ORDINATO
  70 GOSUB 1000:REM ORDINA L'ARRAY
  80 GOSUB 500:REM VISUALIZZA L'ARRAY
     ORDINATO
  90 END
 500 FOR I=0 TO N
 510 PRINT VET(I);" ";
 520 NEXT I
 530 PRINT:PRINT
 540 RETURN
1000 LET PASSO=2*N
1010 LET PASSO=INT(PASSO/2)
1020 IF PASSO>0 THEN GOSUB 2000:GOTO 1010
1030 RETURN
2000 FOR INDICE=0 TO N-PASSO
2020 LET J=INDICE
2030 LET L=J+PASSO
2040 IF VET(L)>VET(J) THEN GOTO 2100
2060 LET M=VET(J)
2070 LET VET(J)=VET(L)
2080 LET VET(L)=M
2090 LET J=J-PASSO:IF J>=0 THEN GOTO 2030
2100 NEXT INDICE
2110 RETURN
```

Questo metodo richiede circa 30 confronti per ordinare 10 elementi, contro i 45 richiesti dal metodo bubble. Con 100 elementi sono invece necessari circa 900 confronti, mentre il metodo bubble ne richiede ben 4950.
Pur senza essere nulla di particolare, la tecnica dello shell sort risulta quindi molto comoda in numerose occasioni, anche in considerazione del fatto che il programma richiede una scarsa occupazione di memoria.
Anche qui, se desideri evitare di inserire manualmente i valori nel vettore, puoi inserire la riga

```
40 LET VET(I)
= INT(RND (0) *5000)
```

Il tuo VIC 20 estrarrà automaticamente dei numeri in modo casuale, assegnandone quindi i valori agli elementi dell'array.

Se invece di un array numerico avessimo voluto ordinare un array alfanumerico, il programma sarebbe stato praticamente identico:

```
   1 PRINT CHR$ (147)
  10 INPUT "QUANTI ELEMENTI";N
  20 DIM VET$(N)
  30 FOR I=0 TO N
  40 GOSUB 3000:REM ESTRAI UNA PAROLA
  50 NEXT I
  60 GOSUB 500:REM VISUALIZZA L'ARRAY
     NON ORDINATO
  70 GOSUB 1000:REM ORDINA L'ARRAY
  80 GOSUB 500:REM VISUALIZZA L'ARRAY
     ORDINATO
  90 END
 500 FOR I=0 TO N
 510 PRINT VET$(I);" ";
 520 NEXT I
 530 PRINT:PRINT
 540 RETURN
1000 LET PASSO=2*N
1010 LET PASSO=INT(PASSO/2)
1020 IF PASSO>0 THEN GOSUB 2000:GOTO 1010
1030 RETURN
2000 FOR INDICE=0 TO N-PASSO
2020 LET J=INDICE
2030 LET L=J+PASSO
2040 IF VET$(L)>VET$(J) THEN GOTO 2100
2060 LET M$=VET$(J)
2070 LET VET$(J)=VET$(L)
2080 LET VET$ (L)=M$
2090 LET J=J—PASSO:IFJ>=0THEN GOTO2030
2100 NEXT INDICE
2110 RETURN
3000 REM ESTRAE UNA STRINGA COMPOSTA
3005 REM DA 4 CARATTERI CASUALI
3010 FOR L=0 TO 3
3020 LET M=INT(RND(0)*91)
3030 IF M<65 THEN GOTO 3020
3040 VET$(I)=VET$(I)+CHR$(M)
3050 NEXT L
3060 RETURN
```

Avremmo cioè dovuto cambiare solo il tipo dell'array: alfanumerico anziché numerico. Per l'assegnazione degli elementi del vettore è stata inserita una subroutine (righe 3000-3060), che estrae in modo casuale delle

stringhe, ciascuna delle quali composta da 4 caratteri. Nel caso tu volessi provare ad inserire manualmente i valori dell'array, dovresti sostituire la riga

```
40 GOSUB 3000
```

con la riga

```
40 INPUT VET$(I)
```

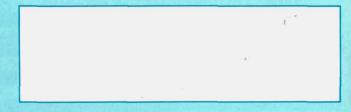
## Teorema di Pitagora

```
10 GOSUB 90
15 ON A GOSUB 30, 50, 75
20 PRINT " PREMI UN TASTO"
25 GOSUB 80 : GOTO 10
30 INPUT "CATETO 1 ="; C1
35 INPUT "CATETO 2 ="; C2
40 LET IP = SQR (C1 ↑ 2 + C2 ↑ 2)
45 PRINT " IPOTENUSA ="; IP : RETURN
50 INPUT "CATETO 1 ="; C1
55 INPUT "IPOTENUSA ="; IP
60 IF IP < C1 THEN GOTO 55
65 LET C2 = SQR (IP ↑ 2 − C1 ↑ 2)
70 PRINT " CATETO 2 ="; C2 : RETURN
75 END
80 GET A$ : IF A$ = "" THEN GOTO 80
85 PRINT "" A$ "" : LET A = VAL (A$) : RETURN
90 PRINT " TEOREMA DI PITAGORA"
95 PRINT " 1 DATI 2 CATETI"
100 PRINT "2 CATETO E IPOTENUSA"
105 PRINT "3 FINE PROGRAMMA"
110 PRINT "QUALE?"; : GOSUB 80
115 IF A < 1 ORA > 3 THEN GOTO 110
120 RETURN
```

Questo semplice programma dimostra la validità dell'uso delle subroutine all'interno di un menù.
Nota alla linea 15 l'istruzione ON A GOSUB ...: in base al valore della variabile A viene richiamata una particolare procedura.

# VIDEOESERCIZI

Annota nell'apposito spazio l'output dei seguenti programmi, poi verifica la soluzione col tuo computer. Se avrai commesso anche un solo errore ripassa la lezione.

```
10 GOSUB 20 : GOSUB 30 : GOSUB 40 : END
20 PRINT "♥ COSA STAMPO CON LA LINEA 40?" : RETURN
30 FOR I = 1 TO 3000 : NEXT : RETURN
40 RETURN : PRINT "QUESTA NON LA VEDI!!!"
```

```
  10 PRINT "♥"
  20 GOSUB 100 : GOSUB 1000
  30 GOSUB 120 : GOSUB 1000
  40 GOSUB 110 : GOSUB 1000
  50 GOSUB 130 : GOSUB 1000
  60 GOSUB 150 : GOSUB 1000
  70 GOSUB 160 : GOSUB 1000
  80 GOSUB 140 : GOSUB 1000
  90 END
 100 PRINT "SONO LA PRIMA A ESSERE STAMPATA." : RETURN
 110 PRINT "... E IO PER TERZA!!!" : RETURN
 120 PRINT "IO SONO CHIAMATA PER SECONDA ..." : RETURN
 130 PRINT "OH! CI SONO ANCH'IO" : RETURN
 140 PRINT "... ED IO L'ULTIMA!" : RETURN
 150 PRINT "IO SONO LA QUINTA ..." : RETURN
 160 PRINT "... IO LA SESTA ..." : RETURN
1000 PRINT "QQ]]]]]]] PREMI UN TASTO" :
1010 GET Z$ : IF Z$ = "" THEN GOTO 1010
1020 PRINT "♥" : RETURN
```

SPONSOR UFFICIALE
FORNITORE UFFICIALE
Azzurra
SFIDA ITALIANA AMERICA'S CUP 1987
RSCG
"Azzurra,
non senti
come mi batte
il cuore?"
In certi casi anche un computer fa i conti con i propri sentimenti, non rimane a guardare freddo e impassibile. Per prepararci alla sfida australiana, seguiamo le prove e gli allenamenti durissimi di Azzurra. Memorizziamo ed elaboriamo i dati digitali ricavati dalle informazioni analogiche sul comportamento della barca in mare. È così che ci sforziamo di rendere ottimali le prestazioni di Azzurra in gara. Honeywell svolge il proprio lavoro come un membro dell'equipaggio: con scrupolo, passione e tanta emozione.
Conoscere e risolvere insieme.
Honeywell
Honeywell Information Systems Italia